

**User Manual for Tactical Language
Identification Software (U)**

DERA/CIS/CIS5/97472B/1.0

19981013 068

EXCLUDED FROM STATESECT A
Approved for public release;
Distribution Unlimited

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE December 1997	3. REPORT TYPE AND DATES COVERED Final Report	
4. TITLE AND SUBTITLE Tactical Language Identification			5. FUNDING NUMBERS F6170896C0003	
6. AUTHOR(S) Dr. P. Nowell, Dr. D.A.Stevens				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Rutherford Appleton Lab Chilton Didcot OX11 0QX UK			8. PERFORMING ORGANIZATION REPORT NUMBER N/A	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) EOARD PSC 802 BOX 14 FPO 09499-0200			10. SPONSORING/MONITORING AGENCY REPORT NUMBER SPC 96-4017	
11. SUPPLEMENTARY NOTES 2 volumes - one is the final report and one is the User Manual for the Tactical Language Identification Software.				
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			12b. DISTRIBUTION CODE A	
13. ABSTRACT (Maximum 200 words) This report results from a contract tasking Rutherford Appleton Lab describes work undertaken into the development of a tactical language identification (TLID) system. Two distinct but complementary approaches have been taken. The first approach is based upon a vector quantizer (VQ) classifier which is able to perform as a language identification system in its own right. The output from the vector quantizer is also used as input for a sequence analysis component that attempts to extract additional structural information from the VQ sequences. Experiments have been performed on the Oregon Graduate Institute (OGI) language identification corpus as well as on a second database supplied by Rome Site. A substantial amount of effort has been devoted to optimizing the performance of the VQ classifier since this forms the basis of all future work. The VQ results on the two databases show that the pre-processing that is applied to the speech signal is important in determining the overall performance. Sequence analysis experiments show that it is also possible to extract additional VQ sequencing information. Results obtained using the VQ show that the performance is highly dependent upon the initial pre-processing that is applied to the speech signal and also to the size of the codebooks. In contrast, the amount of test data which ranges from one second to 45 seconds, does not appear to have a significant effect on the overall results. This suggest that the VQ classifier is best suited to short test utterances where there is little long term information.				
14. SUBJECT TERMS Physics			15. NUMBER OF PAGES 104	
			16. PRICE CODE N/A	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL	

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102

UNCLASSIFIED

Customer Information

Package Title:	Tactical Language Identification
Package Customer:	Rome Laboratory
Package Manager:	Dr. P. Nowell
Research Objective:	Language Identification System
Assignment Number:	E3NMWXXX
Date:	8 th December 1997

**User Manual for Tactical Language
Identification Software (U)**

DERA/CIS/CIS5/97472B/1.0

Cover + iii + 32 pages

8th December 1997

This document is subject to the release conditions printed on the reverse of this page.

DERA

Defence Evaluation
and Research Agency
Farnborough
Hampshire
GU14 6TD

DTIC QUALITY INSPECTED 1

UNCLASSIFIED

DERA is an Agency of the Ministry of Defence

AQ F99-01-0047

Release Conditions

This report has been prepared for Rome Laboratory and, except as indicated, may be used and circulated under the arrangements of contract number:

CSM/6694

It may not, however, be used or copied for any non-Government or commercial purpose without the written agreement of the Defence Evaluation and Research Agency.

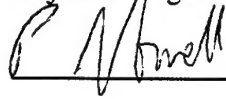
© **Crown Copyright (1997)**
Defence Evaluation and Research Agency (DERA)
Farnborough, Hampshire, GU14 6TD, UK

Approval for copying should be sought from:-
DERA Intellectual Property Department
Tel: 01252 392616

Authorisation

Prepared by: Dr. P. Nowell
Title: Project Manager

Signature:



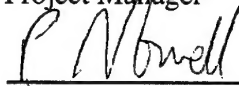
Reviewed by: Dr. R. Series
Title: Project Quality Monitor

Signature:



Authorised by: Dr. P. Nowell
Title: Project Manager

Signature:



Date:

Dec 1997.

Principal Authors

Name: Dr P. Nowell
Appointment: Project Manager
Location: DERA Malvern
Telephone: 01684 896268

Name: Dr D. A. Stevens
Appointment: Researcher
Location: DERA Malvern
Telephone: 01684 894103

Issued by: Dr. P. Nowell
Defence Evaluation and Research Agency (DERA)
St Andrews Road
Malvern
Worcestershire, WR14 3PS.
Telephone 01684-894000.

Record of changes

This is a controlled document.

Additional copies should be obtained through the issuing authority.

In the event of copying locally, each document shall be marked 'Uncontrolled Copy'

Amendment shall be by whole document replacement.

Proposals for change should be forwarded to the issuing authority.

Issue	Date	Details of changes
1.0	8 th December 1997	First Release

List of contents

Authorisation	i
Record of changes.....	ii
1 Introduction	1
1.1 Contractual matters	1
2 Vector Quantiser User Manual	3
2.1 Introduction	3
2.2 External data requirements	3
2.3 Executing the scripts	3
2.4 Changing the control parameters	4
2.5 VQtrain.perl	5
2.6 VQtest.perl	6
2.7 GenSeq.perl	7
2.8 CB3 executable	8
2.9 MCB executable	10
3 Sequence Analysis User Manual	11
3.1 Introduction	11
3.2 External data requirements	11
3.3 Executing the scripts	12
3.4 Changing the control parameters	13
3.5 tlid.rcp	15
3.6 analyse.rcp	16
3.7 cluster.rcp	17
3.8 count.rcp	18
3.9 astrec.rcp	19
3.10 select.rcp	20
3.11 classify.rcp	21
3.12 score.rcp	22
3.13 common.args	23
4 Adapting to new data / languages	25
4.1 Vector quantiser component	25
4.2 Sequence analysis component	25
5 File Locations	27
Initial distribution list	29
Report documentation page	30

UNCLASSIFIED

THIS PAGE INTENTIONALLY BLANK

UNCLASSIFIED

DERA/CIS/CIS5/97472B/1.0

1 Introduction

1.1 Contractual matters

- 1.1.1 This report has been issued by Dr. P. Nowell for Rome Laboratory under contract CSM/6694
- 1.1.2 The report is split into two parts which mirror the major components of the tactical language identification (TLID) software. The first part covers the vector quantiser software whereas the second part covers the sequence analysis software. Further details of the algorithms used and their application to the problem of tactical language identification can be found in the report 'Dr P. Nowell and Dr D. A. Stevens, Final Report on Tactical Language Identification (U), DERA/CIS/CIS5/CR/97472A/1.0, Dec. 1997'.
- 1.1.3 The manual briefly outlines the external data requirements, gives instructions for executing the top level scripts and lists the major control parameters. Further usage instructions are given for the major scripts and executables. Section two contains the user manual for the vector quantiser component of the tactical language identification software. The user manual for the sequence analysis is contained in section three and follows the same format.
- 1.1.4 Section four provides additional information that may be useful in adapting the scripts to new data and/or languages. Finally, section 5 describes the file structure for the installed software.

UNCLASSIFIED

THIS PAGE INTENTIONALLY BLANK

2 Vector Quantiser User Manual

2.1 Introduction

- 2.1.1 This section details the use of the vector quantiser (VQ) based TLID software. Details include descriptions of how to use the three scripts for training and testing of the data as well as for sequence generation as required by the sequence analysis software (see section 3). The guide also includes details of the executables that are at the core of all the VQ scripts.

2.2 External data requirements

- 2.2.1 The external data requirements for the VQ component of the TLID software are listed below.

1. Speechlist files (pairwise list of speech data and annotation files)
2. Speech data files (as listed in the speechlist file)
3. Annotation files (as listed in the speechlist file)
4. Pre-processor definition file (e.g. ppfiles/mfc16+dc+Dc5.pp)

- 2.2.2 The speechlist files are text files containing a pairwise list of the speech data and annotation files that are used for training or testing.

- 2.2.3 The speech data files are binary files containing a spectrogram type representation of the speech data. The first 512 bytes of each file constitutes a header (see appendix A) which describes amongst other things the frame rate and vector size.

- 2.2.4 The annotation files are also binary files and have a similar format to the speech data files (see appendix A). Annotation files are used to specify sub-regions within each speech file that are to be used for training or testing

2.3 Executing the scripts

- 2.3.1 The training process is controlled by a single script (`VQtrain.perl`) which is used repeatedly to generate a codebook model for each target language.

- 2.3.2 Once the language specific codebooks have been generated the VQ language classifier is tested using the script `VQtest.perl`. The program generates an output file containing the scores assigned to each test utterance by each codebook.

- 2.3.3 A third script (`GenSeq.perl`) is used to generate the sequence files used by the sequence analysis software (see section 3). The script uses the codebooks generated by `VQtrain.perl` to convert a series of speech data files into sequence files.

2.4 Changing the control parameters

2.4.1 A number of control parameters are embedded with the training and testing scripts as described below. In addition the two main executables (CB3 and MCB, see sections 2.8 and 2.9 for details) have a number of parameters that can be adjusted to change the training and testing of the vector codebook models.

2.4.2 The major parameters for the training script `VQtrain.perl` are as follows :-

<code>\$MinBkSize</code>	Minimum codebook size to generate
<code>\$MaxBkSize</code>	Maximum codebook size to generate
<code>\$TheTrainProg</code>	Full pathname of cb3 executable
<code>\$TheLanguage</code>	Unique language identifier
<code>\$exptDir</code>	Main root directory for current experiment
<code>\$TheSpeechList</code>	Full pathname of the speechlist file
<code>\$ThePreProc</code>	Full pathname of the pre-processor definition file

2.4.3 The parameters `$MinBkSize` and `$MaxBkSize` are self explanatory in that they are the upper and lower limits of the codebook sizes to generate while training. The main root directory `$exptDir` is the directory where the scripts are held, together with all the sub-directories containing the codebook model files. For any set of experiments where training and testing is used, the `$exptDir` will generally remain the same for all training and testing scripts. The `$TheTrainProg`, `$TheSpeechList` and `$ThePreProc` are described in section on external data requirements (section 2.2).

2.4.4 The test script `VQtest.perl` has a similar set of parameters :-

<code>\$TheTestProg</code>	Full pathname of mcb executable
<code>\$exptDir</code>	Main root directory for current experiment
<code>\$TheSpeechList</code>	Full pathname of the speechlist file
<code>\$ThePreProc</code>	Full pathname of the pre-processor definition file
<code>\$CBList</code>	Full pathname of the codebook list file.

2.4.5 The `$exptDir` and `$ThePreProc` parameters listed above should be identical to those listed in the `VQtrain.perl` script. The `$TheSpeechList` and `$CBList` parameters should point to the lists which identify which files to test and which codebook models to test them with.

2.5 VQtrain.perl

For each language a separate version of this script is required. In general all the parameters will remain identical except for the `$TheLanguage` setting within the script. It is common practice to generate several scripts with the name `VQtrain_${TheLanguage}.perl` to identify which script has generated which language model. For example the script for generating the English model is called `VQtrain_english.perl`.

Usage:

`VQtrain__${TheLanguage}.perl`

Input File(s):

The script requires that the speechlist, speech data files, annotation files and pre-processor file have already been generated (see section 2.2).

Intermediate File(s):

None

Output File(s):

The script generates separate codebook files in the directory `codebooks/$TheLanguage` for each codebook in the range `$MinBkSize` to `$MaxBkSize`. The files are named where `$size` is a zero padded number of states in the codebook. Consequently for a script which trains the English language with `$MinBkSize` set to 4 and `$MaxBkSize` set to 16 there are 3 output files, namely: `english.bk.0004`, `english.bk.0008` and `english.bk.0016`.

`$exptDir/codebooks/$TheLanguage/$Language.bk.$size`

2.6 VQtest.perl

The test script for the VQ based language identification system requires that the `VQtrain.perl` script has been used to generate a set of codebooks for a set of languages.

Usage:

`VQtest.perl`

Input File(s):

The test script requires speechlists, speech data files and annotation files in the same format as that used by the training script.

In addition a codebook list file specifies a list of codebook files and directories such that each line in the codebook list becomes a full pathname description of one of the language specific codebooks when prefixed by the directory `codebooks`. E.g for a sixteen element English codebook the entry would be :-

```
english/english.bk.0016
```

Intermediate File(s):

None

Output File(s):

The output file is contained in the `$exptDir/results` directory and is given the name `results.$size` where `$size` is the number of states used in the codebooks. I.e. the results are written to :-

```
$exptDir/results/results.$size
```

The results file contains a list with three space separated columns. The columns are, in order: the speech data file used in the test; the codebook file against which it has been tested; the score obtained with the given speech file and codebook file. Thus for each speech data listed in the speech list file `N` lines are generated in the results file where `N` is the number of codebooks listed in the codebook list file. For each test file the `N` entries are ordered in preference such that the most likely language is the first of the `N` entries and the least likely codebook is the last entry for that test file.

2.7 GenSeq.perl

GenSeq.perl is used to generate the VQ labelled training and test data for the sequence analysis component of the TLID system (see section 3).

Usage:

GenSeq.perl

Input File(s):

The test script requires speechlists, speech data files and annotation files in the same format as that used by the training script.

In addition a codebook list file specifies a list of codebook files and directories such that each line in the codebook list becomes a full pathname description of one of the language specific codebooks when prefixed by \$exptDir/codebooks. E.g for a sixteen element English codebook the entry would be :-

```
english/english.bk.0016
```

Intermediate File(s):

None

Output File(s):

The output from this script is in the form of sequence files, N per input file as listed in the speechlist file, where N is the number of codebooks listed in the codebook list file. For each speech file in the speechlist a series of sequence files are generated with the same name as the original speech file except that the filename ends in .seq. I.e

```
$exptDir/sequences/<codebook_name>/<speech_filename>.seq
```

The sequence file is a text file of space separated state identifiers with a state number output per frame of input data and each state number being preceded by the letter 's' as a further separator.

2.8 CB3 executable

- 2.8.1 The CB3 executable is designed for both training and testing of a codebook model against a set of speech files as listed in the speechlist file. The options available upon running the CB3 program are as follows together with their default values:

train=+	Train(+) or test(-) mode, valid arguments +,-.
speechlist=	Speech list file name.
preproc=	Pre-processor definition file name
cbfname=	Codebook file name (input if testing, output if training)
distfile=	Output distance filename (debugging)
seqfile=	Output sequence file name descriptor
occ=+	State occupancy flag for calculating distances
minBk=1	Minimum codebook size to generate (power of 2)
maxBk=32	Maximum codebook size to generate (power of 2)
variance=-	Variance terms flag for the codebook model
fullcovar=-	Full covariance flag for the codebook model
globv=-	Global variance flag for the codebook model
min_occ=10	Minimum number of frames per state before splitting
min_var=1e-06	Hard limit on the minimum variance allowable
min_it=10	Minimum number of iterations per codebook size
max_it=30	Maximum number of iterations per codebook size
converge=0.0001	Convergence value
conv_it=3	No. of frames over which convergence is calculated
pmag=1e-06	Perturbation magnification for splitting states
verbose=-	Verbose flag (used for debugging)

- 2.8.2 Within the scripts which use the CB3 program the majority of the default settings are kept. However it is possible to alter the scripts in order to allow different codebook structures or different training conditions. Only one of the three parameters **variance**, **globv** and **fullcovar** may be set to positive although all of them may be set to negative to obtain a system which does not use variance terms at all as in the early OGI experiments [Nowell and Stevens 1997].
- 2.8.3 The **occ=** term is used to include the prior probability of a state occurring in the distance calculation. Any given state will occur a number of times during the training data and as such it is possible to calculate likelihood of this state occurring for a given language and used in the distance measure.
- 2.8.4 For the **seqfile=** parameter the string sequence %s will be replaced by the input speech file name header such that by setting **seqfile=<dir>/%s.seq** it is possible to direct the program to put all the sequence files in <dir> directory, with the same name as the respective speech file in the speech list except for the ending which is replaced by .seq.

- 2.8.5 The `distfile=` parameter is identical to the `seqfile=` parameter except for the format of the file output. The distance file contains information regarding the distance on a per frame basis with one frame per line.

2.9 MCB executable

- 2.9.1 While it is possible to use the CB3 program for testing a file against several codebooks it would be necessary to run the CB program separately for each codebook in turn. The MCB executable is a simple test program which uses many of the same routines as the CB3 program but is able to load multiple codebook files at a time.

cb_dir=	Codebook root directory
cblist=	Codebook list file
speechlist=	Speech list file
preproc=	Pre-processor definition file
results=	Results output file name
verbose=-	Verbose flag (used for debugging)

- 2.9.2 The filenames listed in the **cblist** file are appended to the base directory as defined by the **cb_dir** parameter to give the full path name for each codebook model required. The **speechlist** and **preproc** parameters are used to indicate which speech list file and pre-processor file to use as in the CB3 program. The **results** parameter is location and filename of the output results file.
- 2.9.3 All parameters regarding the codebook structure are unnecessary as they are contained within the codebook file itself and are set up when the file are read in. This is also true for the CB3 program when used in test mode.

3 Sequence Analysis User Manual

3.1 Introduction

- 3.1.1 This section of the report contains the user manual for the sequence analysis component of the tactical language identification (TLID) system. Before any of this software can be used it is first necessary to build the VQ codebooks and then quantise the training and test data. Instructions for doing this are contained in the user manual for the vector quantiser [section 2].

3.2 External data requirements

- 3.2.1 The sequence analysis component of the TLID system builds upon the output of the vector quantiser. It is therefore necessary to first create a suitable quantiser, vector quantise the training and test data and then copy (or create links to) this data in the directory VQdata. The following data needs to be transferred :-

1. Pre-processor definition file (e.g. ppfiles/mfc16+dc+Dc5.pp)
2. Vector quantiser codebooks (e.g. Expts8c/codebooks/english/ english.bk.0128.Z)
3. Vector quantised sequences (e.g. sequences/english.bk.0128/en003stb.seq.Z)
4. Distance matrix (e.g. english.dmat2.0128.Z)
5. Training speechlists (e.g. train_dev_english.tsl)
6. Test speechlists (e.g. test45_dev_english.tsl)
7. Speech data (e.g. /cdrom/dft/en003stb.dft)

- 3.2.2 The pre-processor definition file is a text file which describes the various pre-processing stages that are applied to the speech data. These stages will typically include the calculation of the mean power, cosine coefficients and deltas thereof. For further details of the various pre-processors that have been used and their effects on the performance of the vector quantiser see [Nowell and Stevens 1997].

- 3.2.3 The vector quantiser codebooks are trained on the parameterised speech data. Each codebook has a number of entries (e.g. 128) with means and variances representing prototypical speech vectors. The sequence analysis software requires two codebooks, the first larger codebook is used to model the speech key-fragments and the second smaller codebook (in these experiments containing just 2 entries) is used as a babble model for the non key-fragment speech. The codebooks are stored as compressed files in order to save disk space.

- 3.2.4 The sequences/english.bk.0128/ directory contains the vector quantised training data generated using the larger of the two English codebooks. These files will be processed by the sequence analysis software in order to extract acoustically similar sub-sequences which can then be used by the language classifier.

- 3.2.5 The distance matrix file contains pre-computed distances reflecting the acoustic similarity of the codebook entries in relation to one another and the smaller babble codebook.
- 3.2.6 The training and test speechlists contain a list of the speech files that should be used for training and testing. The training data is split into development and evaluation sub-sets with separate speechlists for each sub-set and language. Likewise, the test data is also split into language specific development and evaluation sub-sets. In addition separate speechlists are used for test files either approximately 45 seconds or 15 seconds long.
- 3.2.7 Finally the raw speech data needs to be on-line. This data can require considerable amounts of disk storage space so, in the case of OGI, the data files for each language are stored on separate CD-ROMs which are accessible over the network. The text file 'files/drives.txt' contains a mapping between the language name and the machine which holds the CD-ROM for that language.

3.3 Executing the scripts

- 3.3.1 The training and testing process is controlled by a single top-level script (`tlid.rcp`) which in turn calls a number of sub-scripts and executables as required for training and testing. To train and test a classifier for the target language '<language>' this script is executed using simply:

```
tlid.rcp <language>
```

- 3.3.2 This script initialises various variables and then goes on to call several sub-scripts which in turn implement the various stages involved in training, testing and scoring the language classifier.

- 3.3.3 During training the following sub-scripts are used

1. analyse.rcp
2. cluster.rcp
3. count.rcp
4. select.rcp

- 3.3.4 and the following are used during testing.

1. count.rcp
2. classify.rcp
3. score.rcp

- 3.3.5 The first sub-script `analyse.rcp` is responsible for extracting similar sub-sequences from the vector quantised training data. These sub-sequences are then clustered by `cluster.rcp` which collects together similar sub-sequences representing the same or similar underlying sounds. Occurrences of each cluster centroid are then counted in the training data by `count.rcp` and these counts are used by `select.rcp` to calculate

'usefulness' scores and determine the most discriminative sub-sequences. During testing occurrences of the most discriminative sub-sequences are counted in the test data again using `count.rcp`. The occurrence counts and associated 'usefulness' scores of each fragment are accumulated and the total used by `classify.rcp` to classify the individual test files. Finally, the classified test files are scored using `score.rcp` and the output is presented as a list of false alarm rates and probability of detection for a range of detection thresholds.

3.4 Changing the control parameters

3.4.1 There are a number of parameters which can be changed in order to modify the behaviour of the sequence analysis software. Most of these are now contained in a single file '`scripts/common.args`' which has comments describing their purpose.

3.4.2 At the end of `tlid.rcp` there is a command

```
$score $basedir $target -4000 250 1000
```

3.4.3 This command runs the scoring script with a range of threshold values ranging from -4000 to 1000 in increments of 250. In order to generate reasonable ROC plots it is necessary to obtain a reasonable number of data points ranging from 0% to 100% false alarms and true detections. This can be achieved by varying the threshold and increment values until the appropriate points are generated. In practice the training and testing scripts are run through to completion for each language and the command

```
scripts/score.rcp . <language> <start> <increment> <end>
```

3.4.4 is repeatedly executed in the experiment directory.

UNCLASSIFIED

THIS PAGE INTENTIONALLY BLANK

3.5 **tlid.rcp**

A single top-level script (`tlid.rcp`) is used to train and evaluate a classifier for each language of interest. This script is called as

Usage:

`tlid.rcp <target>` - Target language (e.g. English)

Input files

None

Intermediate files

See description of component shell scripts

Output files

None

`<target>` is the target language (e.g. german, farsi etc). This script calls a number of other scripts in turn to carry out the various stages involved in training and evaluating the language classifier. These stages and their control scripts are described in order of processing.

3.6 analyse.rcp

The first sub-script `scripts/analyse.rcp` is used to analyse the training speech data files and locate similar sub-sequences (i.e. similar, re-occurring sounds). For the purposes of language identification one would want these sub-sequences to represent phonemes, word fragments (prefixes and suffixes) and possibly entire words (yes, ja etc).

Usage:

```
analyse.rcp <target>      - Target language (e.g. English)
               <codesize>   - Codebook size (e.g. 0128)
               <trainlist>  - File containing list of training files
               <similarity> - Similarity threshold
```

Input File(s):

The script requires that the training speech data has been vector quantised (i.e. converted to a symbol stream). The argument `<trainlist>` gives the name of the file which contains the list of vector quantised training files.

A pre-computed distance matrix is also required for determining the self-similar regions.

Intermediate Files(s):

The processing of each input file leads to the production of an output file (extension `.pms`) containing a list of partial matches.

```
$exptdir/train/$basename.pms
```

The contents of each file are clustered (using `cluster.rcp`) in order to group together similar partial matches, the clusters are stored as intermediate files.

```
$exptdir/train/$basename.cls
```

Output File(s):

The output is a set of files containing a list of similar sub-sequences that were found in the input VQ sequence files.

```
$exptdir/train/$basename.cen
```

These sub-sequences are actually the centroids (i.e. most representative examples) of the clustered partial matches.

3.7 cluster.rcp

The partial match files generated by analyse.rcp typically contain multiple entries for the same underlying sound due to repetitions of the sound in the training file(s). It is necessary to cluster these multiple entries in order to determine a single representative sequence for each sound. Cluster.rcp is called from analyse.rcp to cluster the partial matches within each training file and then again from tlid.rcp to re-cluster the centroids of the clustered training files.

The two stage clustering process is significantly quicker than simply concatenating and clustering the partial matches in all the training files. The end result will be similar provided that the same (or similar) centroids are generated with sufficient frequency in the training data. This assumption will almost certainly be true for any fragments that are likely to be 'useful' since these will occur often and in most training files.

Usage:

```
cluster.rcp <options>      - Additional options
                        <target>      - Target language (e.g. english)
                        <codebook>    - Codebook size (e.g. 0128)
                        <fragments>   - Input file containing fragments
                        <similarity>   - Similarity threshold
                        <limit>       - Minimum cluster size
                        <output>      - Output file
```

Input File(s):

The input consists of a text file (<fragments>) containing the fragments to be clustered and a distance matrix (\$codebook) containing pre-computed inter-state similarity scores.

Intermediate Files(s):

None

Output File(s):

The output is a text file (extension .cls) which contains the individual clusters. The first fragment in each cluster is the centroid (i.e. most representative example) of that cluster.

3.8 count.rcp

Count.rcp is basically a pre-processor for astrec.rcp which actually uses the continuous speech recogniser (which we call astrec) to count the fragment occurrences in the training or test speech.

This shell script generates the necessary control files to configure the speech recogniser to be an acoustic fragment spotter. These include single state babble and fragment models derived from the VQ codebooks, a pronunciation dictionary mapping fragment strings to the appropriate sequence of single state models and syntax files allowing the recognition of acoustic fragments interspersed with 'babble'.

A detailed description of the recogniser configuration files is beyond the scope of this report.

Usage:

```
count.rcp <target>          - Target language (e.g. english)
          <codebook>        - Codebook size (e.g. 0128)
          <speechlist>      - List of VQ'd speech files
          <fragments>       - Fragments to be counted
```

Input File(s):

The input consists of the feature and babble codebooks, top-level recogniser syntax file (files/main.non), a list of fragments to be counted and a file containing a mapping from language to CD-ROM drive containing the corresponding speech data.

Intermediate Files(s):

None

Output File(s):

Recogniser configuration files, the syntax and semantics of these files are beyond the scope of this report.

3.9 astrec.rcp

`Astrec.rcp` is a wrapper which provides the arguments for the speech recognition software. The script first generates a speechlist, using the target language to identify the machine which contains the speech data and thereby the full pathname of each input file. The location of the speechlist and various configuration files are then simply passed on to the recognition software. The output from the recogniser is converted into the format expected by subsequent stages. This involves converting each recogniser `.res` file into a `.cnt` file containing a count reflecting the duration of the file (in this case the number of VQ symbols) followed by a count of the number of occurrences of each fragment.

Usage:

```
astrec.rcp <target>          - Target language (e.g. english)
               <codebook>     - Codebook size (e.g. 0128)
               <speechlist>    - List of VQ'd speech files
               <fragments>     - Fragments to be counted
```

Input File(s):

The input includes the configuration files generated by the previous script (`count.rcp`) as well as the main speechlist file which lists the set of training or test files.

Intermediate Files(s):

The script generates a speechlist for the recogniser which lists the location of each speech file. Separate recognition output files (extension `.res`) are also generated for each input file and these are stored in the directory `$dstdir`. Each file consists of a table containing five columns. The first two columns give the starting and ending frames (where the frame rate is typically 100 frames per second) in the speech signal where the fragment was spotted. The third and fourth columns give the negative log. likelihood score of the detected fragment followed by the average score per frame. Finally, the last column identifies the fragment that was spotted, the number refers to the ranking of the fragment in the input file `$fragments` which contains the list of extracted VQ sequence fragments.

Output File(s):

The output is a set of count files (extension `.cnt`) which, for each recognition results file, contains a count of the total number of VQ symbols in that file (i.e. a measure of the file length) followed by occurrence counts for each fragment in turn.

3.10 select.rcp

Select.rcp uses the .cnt files generated by count.rcp and astrec.rcp to rank and select a subset of the most discriminative fragments. The individual counts for each training file are gathered together and used to generate an intermediate file which contains the total number of occurrences of each fragment per language. The values in this file are used to compute discriminative 'usefulness' scores for each of the fragments. A ranked listing of the fragments and their scores are written to the output file.

Usage:

```
select.rcp <target>          - Target language (e.g. english)
          <fragments>       - Fragments set to select from
          <index>           - Mapping from fragments to counts
          <mode>            - Selection mode (usefulness etc.)
          <selection>       - File to store selection
```

Input File(s):

The input consists of a list of fragments, another list also containing a list of fragments which is used to map the values in the count (.cnt) files back onto the original VQ sub-sequence and finally the directory containing the list of .cnt files.

Intermediate Files(s):

The individual .cnt files are combined into a single file which gives the accumulated counts for each language in turn. The file \$tmpdir/ccounts.tmp consists of a list of languages to be classified, the estimated length of the language specific training data and the number of occurrences of each fragment in each language. These counts are obtained by simply accumulating the counts from the individual training files.

Output File(s):

The output is a single file containing the fragments ranked by their discriminative usefulness scores. The first column is the frequency weighted score which is then followed by the incremental score and finally the VQ sub-sequence to which the scores relate.

3.11 classify.rcp

Classify.rcp is used to classify the .res files according to target language. The input to the process consists of the .cnt files generated by counting occurrences of the selected fragments in the test data and the file containing the list of selected fragments with their discriminative usefulness scores.

The occurrence counts in each .cnt file are normalised by the length of the file and then used along with the incremental 'usefulness' scores to calculate the total score for each file. The output is a file containing the scores for each test file.

Usage:

```

classify.rcp <target>      - Target language (e.g. english)
                  <speechlist> - List of test files
                  <selection>  - Selected fragments
                  <output>     - Language classifications

```

Input File(s):

The input files consist of a speechlist which gives the location of the test files and from which the location of the .cnt files can be determined. A separate file, generated by select.rcp, contains the list of fragments and their associated usefulness scores.

Intermediate Files(s):

None

Output File(s):

The output is a single file which, for each test file, lists the correct language classification, the filename and the accumulated usefulness score. This score should be large when the test file belongs to the target language and small or negative otherwise.

3.12 score.rcp

Usage:

`score.rcp` takes the output file generated by `classify.rcp` and calculates the probability of detection and probability of false alarm for a range of detection thresholds (the actual calculations are actually performed by a perl script). The starting threshold, increment and final threshold value are supplied as arguments. It is often necessary to manually run the scoring script with a variety of different arguments. The data points for the ROC plots displayed previously were generated by manually adjusting the parameters so as to give twenty data points over the range of the plot.

<code>score.rcp</code>	<code><target></code>	- Target language (e.g. english)
	<code><minthreshold></code>	- Initial threshold value
	<code><increment></code>	- Threshold increment
	<code><maxthreshold></code>	- Final threshold value

Input File(s):

Classification file produced by `classify.rcp`

Intermediate Files(s):

None

Output File(s):

None, the results are written directly to the display.

3.13 common.args

Usage:

Common.args is used as a contained for common variables and definitions that are used by a number of other scripts, it is not called in it's own right.

The script defines values for the datatype of the VQ speech files, locations of the VQ symbol set and distance matrix, the DP deletion insertion substitution penalties and the similarity threshold used for the detecting partial matches and clustering.

Input File(s):

None

Intermediate Files(s):

None

Output File(s):

None

UNCLASSIFIED

THIS PAGE INTENTIONALLY BLANK

4 Adapting to new data / languages

4.1 Vector quantiser component

- 4.1.1 The main process in adapting to new data and/or languages lies in the generation of new speech data and annotation files. The speech data files contain a spectrogram type representation of the speech signal and can be generated using either a filterbank or fast fourier transform (FFT) algorithm. The annotation files indicate regions of the speech data files that are to be used for training or testing.
- 4.1.2 Additionally, speechlist files will have to be generated which give the locations of the speech data and annotation files. These files are simple text files which can be most easily generated by modifying existing files such as those included with the installation.

4.2 Sequence analysis component

- 4.2.1 In theory changing the training and test data should require little or no changes to the individual scripts. The codebooks will need to be generated, the speech data vector quantised and then the corresponding files copied to the directory VQdata as described in section 3.2.
- 4.2.2 At the head of `tlid.rcp` there is a string containing the set of languages in the training and test data. If the new set differs from that used for OGI (i.e. english, farsi, french, german, japanese, korean, mandarin, spanish, tamil and vietnamese) then this string will need to be changed.
- 4.2.3 The text file '`files/drives.txt`' contains a mapping between each language and the name of the machine holding the CD-ROM with the speech data for that language. Again, if the language set differs from OGI then this file will need updating.
- 4.2.4 It may also be necessary to optimise the various parameters for the new data set, these parameters are described in section 3.4.

UNCLASSIFIED

THIS PAGE INTENTIONALLY BLANK

5 File Locations

The following file-structure shows the location of various files relative to the top-level directory in which the files are installed. Any changes to this file structure will be recorded in the file README file in the top-level directory.

```

VQdata/          - Contains data from VQ training
  codebooks/      - Contains VQ codebooks
    english/      - Contains english codebooks
      english.bk.0002.Z - Compressed 2 entry codebook
      english.bk.0128.Z - Compressed 128 entry codebook
  ppfiles/        - Contains recogniser pre-processor files
  sequences/      - Contains vector quantised training data
    english.bk.0128/  - Output from 128 element VQ codebook
  speechlists/    - Contain training and test set lists
    train_dev_<language>.tsl - Development training data
    test45_dev_<language>.tsl - 45 second development test
    test10_dev_<language>.tsl - 10 second development test
    test45_eval_<language>.tsl - 45 second evaluation test
    test10_eval_<language>.tsl - 10 second evaluation test

bin/             - Contains various executables
  astrec_2.4.3    - Continuous speech recogniser
  cb2hmm          - Converts codebook to HMM
  ccount_1.0.1    - Collects observation counts
  choose_1.0.6    - Selects most 'Useful' fragments
  dpcluster_1.0.2 - Clusters similar fragments
  expand_1.0.0    - Expands observation counts
  file2file_1.0.3 - Sequence analysis software
  gproc_1.0.0     - Generates recogniser syntax files
  modgen_1.0.0    - Generates recogniser vocabulary file
  uscore_1.0.2    - Calculate 'Usefulness' scores

files/           - Contains various pre-prepared files
  main.non        - Recogniser syntax file
  babble.pre      - Recogniser syntax file
  drives.txt      - Location of CDs

```

```

results/          - Contains experimental results

    <language>/    - Target language (e.g. english)
      train/       - Output from training phase
      test/        - Output from test phase
      tmp/         - Miscellaneous files

scripts/          - Contains shell and perl scripts

    analyse.rcp    - Extract potentially useful fragments
    astrec.rcp     - Count fragment occurrences
    ccounts.rcp    - Accumulate occurrence counts
    classify.rcp    - Classify test data
    cluster.rcp    - Cluster similar fragments
    common.args    - Common arguments
    count1.rcp     - Count fragment occurrences
    gensplist.prl  - Generate speechlist for recogniser
    score.rcp      - Iterates threshold
    sed.cmd        - Commands for 'sed'
    select.rcp     - Select most 'useful' fragments

src/              - Contains source files

    rlabs/         - Source developed for Rome Lab.
      expand.cpp    - Expands observation counts
      gproc.cpp     - Generates recogniser syntax files
      modgen.cpp    - Generates recogniser vocabulary file

test/             - Contains example results

    english/       - Target language (english)
      train/       - Output from training phase
      test/        - Output from test phase
      tmp/         - Miscellaneous files

```

A. Binary File Headers

- A.1.1 The vector quantiser software operates on binary speech data and annotation files which will need to be generated if the software is to be used on new data. Each binary data file has a header which describes the data that follows. The header is identified by the characters 'SRUHEAD0' and contains a number of fields as shown below.

char ident[8];	- header identifier
int32 byt_per_frame;	- Number of bytes per frame
int32 file_type;	- File type
int32 data_type;	- Data type
int32 res_len;	- Not used
int32 data_len;	- Length of data
int32 samplerate;	- Sampling rate
int32 downsample;	- Downsample rate
float max_val;	- Maximum data value
float max_scale;	- Maximum data scale
float min_val;	- Minimum data value
float min_scale;	- Minimum data value
int32 no_coms;	- Number of comments
int32 com_len;	- Length of comments
int32 ex_head_len;	- Length of extra data
int32 pad_len;	- Length of padding data
int32 offset;	- Offset relative to signal file
int32 machine;	- machine formats flag
int32 reserved;	- reserved

- A.1.2 The meanings of those fields that are important for subsequent software and typical values for speech data and annotation files are given in appendices A.1 and A.2.

A.2 Typical speech data file header

- A.2.1 The following represents a typical header from a speech data file :-

identifier	SRUHEAD0
byte per frame	80
file type	4 (filter bank log/transformed)
data type	11 (SRUbank coded log powers)
data len	398240
samplerate	8000
downsampled	-80
offset	0
max_val	-1.000000
max_scale	1.000000
min_val	1.000000
min_scale	1.000000
number of comments	4
comment length	176
extra header length	0
padding length	256
architecture bits	0x311 (DECstation, little-endian, IEEE float 32)

- A.2.2 It can be seen from the header that the data in this file consists of log. filterbank coefficients (filetype 4), and the coefficients are stored as unsigned characters scaled in 0.5dB steps (data type 11). The original speech was sampled at 8kHz but the frame rate of the coefficients is 100Hz hence the value of the downsampled field. The total size of the filterbank coefficients is 398240 bytes (this corresponds to 4978 frames and 62.2 seconds of speech).
- A.2.3 There are four comments (not displayed) which are contained in 176 bytes following the header and there is no extra header information. In order for the total size of the header and comments to be a multiple of 512 bytes it was necessary to append 356 padding characters.
- A.2.4 The remaining fields such as maximum and minimum values are not used at present and were not computed.

A.3 Typical annotation file header

- A.3.1 The following represents a typical annotation file header :-

identifier	SRUHEAD0
byte per frame	76
file type	12 (annotation)
data type	22 (Sentence annotation)
data len	2660
samplerate	19980
downsampled	-1
offset	0
max_val	3546.000000
max_scale	1.000000
min_val	94.000000
min_scale	1.000000
number of comments	6
comment length	268
extra header length	224
padding length	452
architecture bits	0x12 (presumed VAX, little-endian, VAX float 32)

- A.3.2 The header shows that this file contains annotation data (file type 12) at the sentence level (data type 22). The annotation data is stored in fixed width 'frames' which in this case are 76 bytes long. The total data length is 2660 bytes corresponding to 35 annotations. The signal data was sampled at 19.98 kHz and the 'start' and 'end' values of the annotation tags also refer to the same sampling rate (downsampled = -1).
- A.3.3 The remaining fields are as described for the previous header. Each annotation tag is stored as two 32 bit integers for the start and end points followed by annotation text. The total length of the integers and text should equal `byt_per_frame`.

Initial distribution list

Name	Organisation	Copy
John Parker Jim Cupples	Rome Laboratory / IRAA 32 Hangar Road Rome, NY 13441-4114	1-2
Dr. M.F. Levy	Signal Science Limited 5 Hutchcombe Farm Close Oxford, OX2 9HG	3-4
Audrey James	SRU Librarian DERA Malvern Worcs, WR14 3PS	5-8

Report documentation page

Originator's report number: DERA/CIS/CIS5/97472B/1.0

Originator's name and location: Issued by Dr. P. Nowell.
Defence Evaluation and Research Agency, St Andrews Road, Malvern, Worcestershire, WR14 3PS.
Telephone 01684-894000.

Contract and period covered: CSM / 6694, [1996-1997]

Customer: Rome Laboratory

Report protective marking and descriptor: UNCLASSIFIED

Date of issue: December 1997

Number of pages: Cover + iii + 32 pages

Number of references: 0 Reference(s)

Title: User Manual for Tactical Language Identification Software

Translation/conference details: None

Title classification: Unclassified

Author(s): Dr P. Nowell and Dr D. A. Stevens

Key words: Tactical Language Identification, User Manual